Master Informatique

Projet LRC
Ecriture en Prolog d'un démonstrateur basé sur l'algorithme des tableaux pour la logique de description ALC

Étudiants
Basci Onur
Mermoz Hatem

December 20, 2023

Contents

I Description du répertoire

Le fichier principal main.py contient l'essence du programme, fournissant ainsi les réponses aux trois sections mentionnées. Les fichiers t_abox contiennent différents types de tbox et abox pour tester divers scénarios. Le fichier tests.pl contient des cas de test permettant de vérifier les prédicats individuellement.

II Execution du Programme

Pour executer le programme essentiel, il faut dabord charger le fichier main.pl par la commande "[main].". En suite il suffit seulement d'éxecuter la commande main_programme qui commence le programme.
Le programme procède ensuite à l'exécution de la partie 1, où il présente les tbox et abox sous forme normale négative, avec les éléments complexes remplacés par leurs équivalences. Ensuite, le programme exécute la partie 2, demandant à l'utilisateur d'ajouter une proposition de type $I : C$ ou de type $C1 \sqcap C2 \sqsubseteq \bot$. Pour ajouter une proposition de type 1, l'utilisateur doit saisir "1.", et pour une proposition de type 2, "2.". Selon le choix de l'utilisateur, il doit ensuite saisir soit une instance et un concept, soit deux concepts provenant d'un fichier t_abox.

Enfin, le programme exécute la partie 3 du projet, affichant toutes les étapes qu'il accomplit. À chaque étape, il présente également la situation actuelle de l'Abox en affichant Lie, Lpt, Li, Lu, Ls. En conclusion, le programme indique si la proposition saisie a été démontrée. Il affiche 'All leaves are closed: The proposition has been proven.' si la proposition est démontrée et 'There is an open leaf: The proposition could not be proven.' sinon.

```
Normal TBox: [(auteur,and(personne,some(aEcrit,livre))),(editeur,and(personne,and(not(some(
aEcrit,livre)),some(aEdite,livre)))),(parent,and(personne,some(aEnfant,anything))),(sculpte
ur,and(personne,some(aCree,sculpture)))]
Normal Abox Individuals: [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,
livre),(vinci,personne)]
Normal Abox Relationships: [(michelAnge,david,aCree),(michelAnge,sonnets,aEcrit),(vinci,joc
onde,aCree)][LOG] Checking TBox syntax...
[LOG] TBox syntax verification successful
[LOG] Checking ABox syntax...
[LOG] ABox syntax verification successful
[LOG] No self-references in TBox
[LOG] Processing Boxes...
[LOG] Transformation completed

Enter the number of the type of proposal you wish to demonstrate:
1 A given instance belongs to a given concept.
2 Two concepts have no elements in common (they have an empty intersection).
|: 1.
Entrez I :
|: michelAnge.
Entrez C :
|: sculpteur.

Abox Individuals:
[(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personne),(
michelAnge,or(not(personne),all(aCree,not(sculpture))))]
Abox Relationships:
[(michelAnge,david,aCree),(michelAnge,sonnets,aEcrit),(vinci,joconde,aCree)]==============
======


Règle or

current state

Lie: []

Lpt: []

Li: []

Lu: []

Ls: [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personn
e),(michelAnge,not(personne))]

current state

Lie: []

Lpt: [(michelAnge,all(aCree,not(sculpture)))]

Li: []

Lu: []

Ls: [(david,sculpture),(joconde,objet),(michelAnge,personne),(sonnets,livre),(vinci,personn
```

Figure 1: Un exemple d'éxecution

III predicats

III.1 Predicats utilitaires

On commence par expliquer les prédicats utilitaires qu'on s'est servi plusieurs fois durant le projet.

```
%PART 1
%predicats utils
concatenate([], L, L).
concatenate([X|Rest1], L2, [X|Result]) :-
    concatenate(Rest1, L2, Result).

delete_element(_, [], []).

delete_element(Element, [Element | Tail], Tail).

delete_element(Element, [Head | Tail], [Head | ResultTail]) :-
    delete_element(Element, Tail, ResultTail).

% Predicate to extract substring until the first opening parenthesis
get_type(Term, Substring) :-
    term_to_atom(Term, Atom),  % Convert the term to an atom
    atom_chars(Atom, AtomChars),  % Convert the atom to a list of characters
    get_substring_chars(AtomChars, SubstringChars),  % Get the substring characters
    atom_chars(Substring, SubstringChars).  % Convert the substring characters back to an atom

% Base case: Empty list
get_substring_chars([], []).

% Stop when an opening parenthesis is encountered
get_substring_chars(['('|_], []) :- !.

% Add the current character to the substring
get_substring_chars([Char|Rest], [Char|Substring]) :-
    get_substring_chars(Rest, Substring).
```

Figure 2: prédicats utilitaires

Le prédicat **concatenate** nous permet de concaténer deux listes en une seule liste. Le prédicat **delete_element** crée une liste contenant tous les éléments de la liste donnée, sauf l'élément à effacer. Le prédicat **get_type** nous permet de trouver le type d'un concept, c'est-à-dire la sous-séquence allant jusqu'à la première apparition de la parenthèse ouvrante. Par exemple, si Element = and(C1, C2), alors Substring prend la valeur "and".

III.2 Partie 1

```
concept(C) :- cnamea(C).
concept(C) :- cnamena(C).

% On vérifie la grammaire de la logique ALC
concept(not(C)) :- concept(C).
concept(or(C1, C2)) :- concept(C1), concept(C2).
concept(and(C1, C2)) :- concept(C1), concept(C2).
concept(all(R, C)) :- rname(R), concept(C).
concept(some(R, C)) :- rname(R), concept(C).
```

Figure 3: Concept

Un terme est un concept s'il est un concept atomique, non atomique ou une combinaison des concepts atomiques ou non atomiques par les opérations not, and, or, some et all.

```
% Pour la Tbox
definitionT(CNA, CNA2) :- cnamena(CNA), concept(CNA2).
verif_Tbox([(CNA, CNA2) | L]) :-
    definitionT(CNA, CNA2),
    verif_Tbox(L).
verif_Tbox([]).
```

Figure 4: Vérification de Tbox

Pour vérifier si un Tbox est bien défini, on examine chaque élément pour s'assurer qu'il s'agit d'un tuple où le premier élément est un concept non atomique et le deuxième est un concept quelconque. Cette vérification s'effectue de manière récursive.

```
%pour la Abox
verif_AboxI([(I, C) | L]) :-
    inst(I,C),
    verif_AboxI(L).
verif_AboxI([]).

verif_AboxR([(I1, I2, R) | L]) :-
    instR(I1, I2, R),
    verif_AboxR(L).
verif_AboxR([]).

verif_Abox(Abi, Abr) :-
    verif_AboxI(Abi),
    verif_AboxR(Abr).
```

Figure 5: Vérification de Abox

On vérifie de manière similaire la Abox d'instances et la Abox de relations. On effectue une vérification récursive pour s'assurer que chaque élément des Abox est une instance.

```
% Vérification d'auto réference, on garde toutes les élements non
% atomique dans une liste et on vérifie si on rencontre 2 fois cet
% élement. Si c'est le cas alors autoref est vrai.
% L c'est pour obtenir les élements rencontrés

verif_Autoref([C | L]) :-
    equiv(C, E),
    (autoref(C, E, []);
    verif_Autoref(L)).

autoref(C, not(C2), L) :-
    autoref(C, C2, L).

autoref(C, and(C1, C2), L) :-
    autoref(C, C1, L); autoref(C, C2, L).

autoref(C, or(C1, C2), L) :-
    autoref(C, C1, L), autoref(C, C2, L).

autoref(C, all(_, C2), L) :-
    autoref(C, C2, L).

autoref(C, some(_, C2), L) :-
    autoref(C, C2, L).

autoref(C, C1, L) :-
    member(C, L);
    cnamena(C1), concatenate(L, [C1], L2), equiv(C1, Z), autoref(C, Z, L2).

pas_autoref(C, CG, L) :- \+ autoref(C, CG, L).
```

Figure 6: Vérification d'autoréférence

Pour vérifier l'autoréférence dans l'Abox, on utilise une liste pour analyser chaque concept. On remplace chaque élément par son équivalence (si cela existe), et si on rencontre un élément deux fois, alors on conclut qu'il y a autoréférencement. Si on peut analyser jusqu'à la fin de la proposition, cela signifie qu'il n'y a pas d'autoréférencement, et donc le prédicat est faux.

```
nnf(not(and(C1,C2)),or(NC1,NC2)):- nnf(not(C1),NC1),nnf(not(C2),NC2),!.
nnf(not(or(C1,C2)),and(NC1,NC2)):- nnf(not(C1),NC1),
nnf(not(C2),NC2),!.
nnf(not(all(R,C)),some(R,NC)):- nnf(not(C),NC),!.
nnf(not(some(R,C)),all(R,NC)):- nnf(not(C),NC),!.
nnf(not(not(X)),Y):- nnf(X,Y),!.
nnf(not(X),not(X)):-!.
nnf(and(C1,C2),and(NC1,NC2)):- nnf(C1,NC1),nnf(C2,NC2),!.
nnf(or(C1,C2),or(NC1,NC2)):- nnf(C1,NC1), nnf(C2,NC2),!.
nnf(some(R,C),some(R,NC)):- nnf(C,NC),!.
nnf(all(R,C),all(R,NC)) :- nnf(C,NC),!.
nnf(X,X).
```

Figure 7: Mise en forme normale négative

Comme indiqué dans l'énoncé, le prédicat nnf met une formule dans une forme normale négative.

```
atom_concate5(A1, A2, A3, A4, A5, X) :-
    atom_concat(A1, A2, Temp1),
    atom_concat(Temp1, A3, Temp2),
    atom_concat(Temp2, A4, Temp3),
    atom_concat(Temp3, A5, X).

atom_concate3(A1, A2, A3, X) :-
    atom_concat(A1, A2, Temp1),
    atom_concat(Temp1, A3, X).
```

Figure 8: Concatener les chaines de caractères

Les prédicats **atom_concatenate5** et **atom_concatenate3** nous permettent de concaténer respectivement 5 et 3 chaînes de caractères. Ces prédicats sont utilisés pour simplifier les prédicats suivants.

```
forme_atomique(and(CG1, CG2), X) :-
    forme_atomique(CG1,S1),forme_atomique(CG2, S2),
    term_string(S1, S_1), term_string(S2, S_2),
    atom_concate5("and(", S_1,"," ,S_2,")", X1),
    term_string(X, X1).


forme_atomique(or(CG1, CG2), X) :-
    forme_atomique(CG1,S1),forme_atomique(CG2, S2),
    term_string(S1, S_1), term_string(S2, S_2),
    atom_concate5("or(", S_1,"," ,S_2,")", X1),
    term_string(X, X1).

forme_atomique(all(R, CG), X) :-
    forme_atomique(CG,S1),
    term_string(S1, S_1), term_string(R, S_2),
    atom_concate5("all(", S_2,"," ,S_1,")", X1),
    term_string(X, X1).

forme_atomique(some(R, CG), X) :-
    forme_atomique(CG,S1),
    term_string(S1, S_1), term_string(R, S_2),
    atom_concate5("some(", S_2,"," ,S_1,")", X1),
    term_string(X, X1).

forme_atomique(not(CG), X) :-
    forme_atomique(CG,S1),
    term_string(S1, S_1),
    atom_concate3("not(",S_1,")" , X1),
    term_string(X, X1).


forme_atomique(CG, X) :-
    cnamena(CG), equiv(CG, X); X = CG.
```

Figure 9: Mettre sous forme atomique

Le prédicat **forme_atomique** met les formules en forme atomique. Cela signifie que si un concept non atomique a une équivalence, alors on le remplace par son équivalence.

```
% CG concept complex, X resultat sous forme normale négatif avec des
% élements atomique
traitement_elem_complex(CG, X) :-
    forme_atomique(CG, Y),
    nnf(Y, X).

traitement_Tbox([], Init, L) :- L = Init.
traitement_Tbox([(CA, CG) | L], Init, Lfinal) :-
    pas_autoref(CA, CG, []), %check if there is not autoref
    traitement_elem_complex(CG, Y),
    concatenate(Init, [(CA, Y)], Lc),
    traitement_Tbox(L, Lc, Lfinal).


% cela marche de la même manière que Tbox comme les element de Abox
% d'instance est composé de (instance, element complex)
traitement_AboxI([], Init, L) :- L = Init.
traitement_AboxI([(CA, CG) | L], Init, Lfinal) :-
    pas_autoref(CA, CG, []), %check if there is not autoref
    traitement_elem_complex(CG, Y),
    concatenate(Init, [(CA, Y)], Lc),
    traitement_AboxI(L, Lc, Lfinal).


%PART 2
```

Figure 10: Traitment Tbox et Abox

Pour traiter un concept non atomique, on remplace d'abord toutes les équivalences dans la formule, puis on met la formule dans sa forme normale négative. Le prédicat **traitement_elem_complex** réalise ce traitement pour un seul élément. Le prédicat **traitement_Tbox** effectue ce traitement de manière récursive pour chaque élément de la Tbox. De même pour le prédicat **traitement_AboxI**.

III.3 Partie 2

```
saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox) :-
    nl,write('Enter the number of the type of proposal you wish to demonstrate:'),nl,
    write('1 A given instance belongs to a given concept.'),nl,
    write('2 Two concepts have no elements in common (they have an empty intersection).'),nl, read(R), suite(R,Abi,Abi1,Tbox).

suite(1,Abi,Abi1,Tbox) :-
    acquisition_prop_type1(Abi,Abi1,Tbox),!.

suite(2,Abi,Abi1,Tbox) :-
    acquisition_prop_type2(Abi,Abi1,Tbox),!.

suite(R,Abi,Abi1,Tbox) :-
    nl,write('This answer is incorrect.'),nl,
    saisie_et_traitement_prop_a_demontrer(Abi,Abi1,Tbox).

get_prop1(I, C) :-
    write('Entrez I : '), nl,
    read(I),
    write('Entrez C : '), nl,
    read(C),
    instC(I, C).

get_prop2(C1, C2) :-
    write('Entrez C1 : '), nl,
    read(C1),
    write('Entrez C2 : '), nl,
    read(C2),
    concept(C1), concept(C2).
```

Figure 11: Saisir Proposition

Les prédicats **get_prop1** et **get_prop2** nous permettent de récupérer les propositions de type 1 et 2. On vérifie également si les formules sont des instances de concept ou bien des intersections de concepts.

```
%on suppose que le Tbox est déja traité
acquisition_prop_type1(Abi, Abi1, Tbox) :-
    get_prop1(I, C),
    traitement_elem_complex(not(C), Y),
    concatenate(Abi, [(I, Y)], Abi1).


generer_random_Iname(RI) :-
    random(0,100000, RN),
    atom_concat('i', RN, RIS),
    term_string(RI, RIS).


acquisition_prop_type2(Abi,Abi1,Tbox) :-
    generer_random_Iname(RI),
    get_prop2(C1, C2),
    traitement_elem_complex(not(and(C1, C2)), Y),
    concatenate(Abi, [(RI,Y)], Abi1).
```

Figure 12: Acquisition de propositions

III.4 Partie 3

Les prédicats **acquisition_prop_type1** et **acquisition_prop_type2** nous permettent d'ajouter les propositions saisies dans l'Abox. Ils placent d'abord la négation de la formule en forme atomique et également en forme normale négative.

```
tri_Abox(Abi, Lie, Lpt, Li, Lu, Ls) :-
    get_Lie(Abi, [], Lie),
    get_Lpt(Abi, [], Lpt),
    get_Li(Abi, [], Li),
    get_Lu(Abi, [], Lu),
    get_Ls(Abi, [], Ls, Lie, Lpt, Li, Lu).
```

```
get_Lie([], Init,Lie) :- Lie = Init.
get_Lie([(I, some(R, C)) | L], Init, Lie) :-
    concatenate(Init, [(I, some(R, C))], Y),
    get_Lie(L, Y, Lie).


get_Lie([(_, _) | L], Init, Lie) :-
    get_Lie(L, Init, Lie).


get_Lpt([], Init,Lpt) :- Lpt = Init.
get_Lpt([(I, all(R, C)) | L], Init, Lpt) :-
    concatenate(Init, [(I, all(R, C))], Y),
    get_Lpt(L, Y, Lpt).


get_Lpt([(_, _) | L], Init, Lpt) :-
    get_Lpt(L, Init, Lpt).


get_Li([], Init,Li) :- Li = Init.
get_Li([(I, and(C1, C2)) | L], Init, Li) :-
    concatenate(Init, [(I, and(C1, C2))], Y),
    get_Li(L, Y, Li).


get_Li([(_, _) | L], Init, Li) :-
    get_Li(L, Init, Li).


get_Lu([], Init,Lu) :- Lu = Init.
get_Lu([(I, or(C1, C2)) | L], Init, Lu) :-
    concatenate(Init, [(I, or(C1, C2))], Y),
    get_Lu(L, Y, Lu).


get_Lu([(_, _) | L], Init, Lu) :-
    get_Lu(L, Init, Lu).
```

Le prédicats **get_Li, get_Lu ...** nous permet d'obtenir des listes concept commeçant par some, or... à partir de Abox. En utilisant les prédicats de "get", le prédicat **tri_Abox** nous génère les listes suivantes:

- la liste Lie des assertions du type (I,some(R,C))
- la liste Lpt des assertions du type (I,all(R,C))
- la liste Li des assertions du type (I,and(C1,C2))
- la liste Lu des assertions du type (I,or(C1,C2))
- a liste Ls des assertions restantes, à savoir les assertions du type (I,C) ou (I,not(C)), C étant un concept atomique.

```
get_Ls([], Init, Ls, Lie, Lpt, Li, Lu) :- Ls = Init.
get_Ls([(I, C) | L], Init, Ls, Lie, Lpt, Li, Lu) :-
    ((member((I, C), Lie);
    member((I, C), Lpt);
    member((I, C), Li);
    member((I, C), Lu)),
    get_Ls(L, Init, Ls, Lie, Lpt, Li, Lu)); %soit C appartient à Lie, Lpt, Li ou Lu dans ce cas là on passe au prochain term.
    concatenate(Init, [(I, C)], Y), %sinon on l'ajoute dans Ls
    get_Ls(L, Y, Ls, Lie, Lpt, Li, Lu).
```

Figure 13: Tri Abox

```
evolue((I, and(C1, C2)), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1) :-
    evolue_single((I, C1), Lie, Lpt, Li, Lu, Ls, Liei, Lpti, Lii, Lui, Lsi),
    evolue_single((I, C2), Liei, Lpti, Lii, Lui, Lsi, Lie1, Lpt1, Li1, Lu1, Ls1).


evolue((I, C), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1) :-
    evolue_single((I, C), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1).


evolue_single((I, C), Lie, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1) :-
    (get_type(C, T), T == and,
    Lie1 = Lie, Lpt1 = Lpt, Lu1 = Lu, Ls1 = Ls,
    concatenate(Li, [(I, C)], Li1));
    (get_type(C, T), T == or,
    Lie1 = Lie, Lpt1 = Lpt, Li1 = Li, Ls1 = Ls,
    concatenate(Lu, [(I, C)], Lu1));
    (get_type(C, T), T == some,
    Li1 = Li, Lpt1 = Lpt, Lu1 = Lu, Ls1 = Ls,
    concatenate(Lie, [(I, C)], Lie1));
    (get_type(C, T), T == all,
    Lie1 = Lie, Li1 = Li, Lu1 = Lu, Ls1 = Ls,
    concatenate(Lpt, [(I, C)], Lpt1));
    (get_type(C, T), T == not,
    Lie1 = Lie, Li1 = Li, Lu1 = Lu, Lpt1 = Lpt,
    concatenate(Ls, [(I, C)], Ls1));
    (cnamea(C),
    Lie1 = Lie, Lpt1 = Lpt, Lu1 = Lu, Li1 = Li,
    concatenate(Ls, [(I, C)], Ls1)).
```

Figure 14: evolue

A représente une nouvelle assertion de concepts à intégrer dans l'une des listes Lie, Lpt, Li, Lu ou Ls qui décrivent les assertions de concepts de la Abox étendue et Lie1, Lpt1, Li1,Lu1 et Ls1 représentent les nouvelles listes mises à jour.

```
test_clash([(I, C) | L], Li) :-
    member((I, not(C)), Li); %not(C) est dans la liste d'instance, Clash
    test_clash(L, Li).
```

Figure 15: Test clash

Le prédicat **test_clash** vérifie s'il y a un conflit dans une Abox. Si l'Abox contient à la fois un élément et sa négation, alors le prédicat est vrai.

```
complete_some([(A, some(R, C)) | L], Lpt, Li, Lu, Ls, Abr) :-
    nl,write('Règle il existe'), nl,
    generer_random_Iname(B), generer_random_Iname(Buffer),
    nl, write('before some '), write(Buffer),nl,
    concatenate([(A, some(R, C))], L, Lie),
    print_current_states(Lie, Lpt, Li, Lu, Ls),
    evolue((B, C), L, Lpt, Li, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
    concatenate(Abr, [(A, B, R)], Abr1), %mettre à jour Abr
    nl, write('after some'), write(Buffer), nl,
    print_current_states(Lie1, Lpt1, Li1, Lu1, Ls1),
    (test_clash(Ls1, Ls1); %soit il y a un clash et on arête
    resolution(Lie1, Lpt1, Li1, Lu1, Ls1, Abr1) %sinon on continue la résolution
    ).
```

Figure 16: Complete some

Ce prédicat cherche une assertion de concept de la forme $(I, \text{some}(R, C))$ dans la liste Lie. S'il en trouve une, il cherche à appliquer la règle $\exists$ (voir le schéma de la boucle de contrôle présenté plus haut, qui permet de comprendre la structure de ce prédicat).

```
transformation_and(Lie, Lpt, [(A, and(C1, C2)) | L], Lu, Ls, Abr) :-
    nl,write('Règle et'), nl,
    concatenate([(A, and(C1, C2))], L, Li),
    generer_random_Iname(Buffer),
    nl, write('before and'), write(Buffer), nl,
    print_current_states(Lie, Lpt, Li, Lu, Ls),
    evolue((A, and(C1, C2)) , Lie, Lpt, L, Lu, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
    nl, write('after and'),  write(Buffer), nl,
    print_current_states(Lie1, Lpt1, Li1, Lu1, Ls1),
    (test_clash(Ls1, Ls1); %soit il y a un clash et on arête
    resolution(Lie1, Lpt1, Li1, Lu1, Ls1, Abr) %sinon on continue la résolution
    ).
```

Figure 17: Transformation and

Ce prédicat cherche une assertion de concept de la forme $(I, \text{and}(C1, C2))$ dans la liste Li. S'il en trouve une, il cherche à appliquer la règle $\sqcap$ (voir le schéma de la boucle de contrôle présenté plus haut, qui permet de comprendre la structure de ce prédicat).

```
transformation_or(Lie, Lpt, Li, [(A, or(C1, C2)) | L], Ls, Abr) :-
    nl,write('Règle or'), nl,
    (evolue((A, C1) , Lie, Lpt, Li, L, Ls, Lie1, Lpt1, Li1, Lu1, Ls1),
     print_current_states(Lie1, Lpt1, Li1, Lu1, Ls1),
    (test_clash(Ls1, Ls1); %soit il y a un clash et on arête
    resolution(Lie1, Lpt1, Li1, Lu1, Ls1, Abr) %sinon on continue la résolution
    ),%premier branche
    evolue((A, C2) , Lie, Lpt, Li, L, Ls, Lie2, Lpt2, Li2, Lu2, Ls2),
    print_current_states(Lie2, Lpt2, Li2, Lu2, Ls2),
    (test_clash(Ls2, Ls2); %soit il y a un clash et on arête
    resolution(Lie2, Lpt2, Li2, Lu2, Ls2, Abr) %sinon on continue la résolution
    )).%second branche
```

Figure 18: Transformation and

Ce prédicat cherche une assertion de concept de la forme $(I, \text{or}(C1, C2))$ dans la liste Lu. S'il en trouve une, il cherche à appliquer la règle $\sqcup$ (voir le schéma de la boucle de contrôle présenté plus haut, qui permet de comprendre la structure de ce prédicat).

```
deduction_all(Lie, [(A, all(R, C)) | L], Li,Lu,Ls,Abr) :-
    nl,write('Règle pour tout'), nl,
    get_a_b_R(A, R, Abr, [], La_b_R), %list contenant les elements de types (a, b) : R
    put_bc_for_all(C, La_b_R, L, Ls1),
    print_current_states(Lie, L, Li, Lu, Ls1),
    (test_clash(Ls1, Ls1); %soit il y a un clash et on arête
    resolution(Lie, L, Li, Lu, Ls1, br) %sinon on continue la résolution
    ).

%ce prédicat nous donne toutes les instances de type (a,b):R
get_a_b_R(_, _, [], Init, Lfinal) :- Lfinal = Init.
get_a_b_R(A, R,[(A1, B1, R1) | L], Init, Lfinal) :-
    A == A1, R == R1, %soit A, B, R est dans la liste
    concatenate(Init, [(A,B1,R)], L1),
    get_a_b_R(A, R, L, L1, Lfinal);
    get_a_b_R(A, R, L, Init, Lfinal). % sinon on vérifie le prochain élement

%ce prédicat ajoute b: C pour chaque element de type (a, b, R)
put_bc_for_all(_, [], Ls, L_final) :- L_final = Ls.
put_bc_for_all(C, [(_, B, _) | L], Ls, Ls_final) :-
    evolue((B, C), L, _, _, _, Ls, _, _, _, _, Ls1),
    put_bc_for_all(C, L, Ls1, Ls_final).
```

Figure 19: Deduction All

Ce prédicat cherche une assertion de concept de la forme $(I, \text{all}(R, C))$ dans la liste Lpt. S'il en trouve une, il cherche à appliquer la règle $\forall$ (voir le schéma de la boucle de contrôle présenté plus haut, qui permet de comprendre la structure de ce prédicat).

```
resolution(Lie, Lpt, Li, Lu, Ls, Abr) :-
    complete_some(Lie, Lpt, Li, Lu, Ls, Abr);
    transformation_and(Lie, Lpt, Li, Lu, Ls, Abr);
    deduction_all(Lie, Lpt, Li, Lu, Ls, Abr);
    transformation_or(Lie, Lpt, Li, Lu, Ls, Abr).
```

Figure 20: Résolution

Le prédicat **resolution** a pour paramètres les 5 listes Lie, Lpt, Li, Lu, Ls, des assertions de concepts de la Abox étendue et la liste Abr des assertions de rôles de celle-ci. Ce prédicat va utiliser les prédicats suivants, qui possèdent tous les mêmes paramètres qui traduisent l'état de la Abox étendue à un instant donné :

IV Tests

Pour tester les prédicats individuellement, vous pouvez trouver les lignes de commande à exécuter pour chaque prédicat dans le fichier tests.pl.

V Problèmes rencontrés

Nous avons observé que, dans certains cas de Tbox/Abox et pour certaines commandes saisies, le programme entre dans une boucle infinie. Malheureusement, nous n'avons pas eu le temps de résoudre ce problème.